

N 71 1 1 3 1 4
CR 102913

MCR-70-38
(Supplement 1)

5
COPY NO. _____

CASE FILE
COPY

FINAL REPORT
FOR
SUPPLEMENT ONE
FORMULATION
OF A
TELEMETRY COMPUTER PROGRAM
CONTRACT NAS8-24017
SEPTEMBER 1970

Prepared For

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER ALABAMA 35812

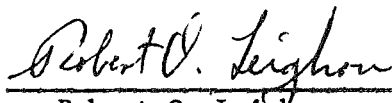
Electronics Research Department
Martin Marietta Corporation
P. O. Box 179, Denver, Colorado 80201

MCR-70-38
(Supplement 1)

FINAL REPORT
FOR
SUPPLEMENT ONE
FORMULATION
OF A
~~THERM~~ THERMOMETRY COMPUTER PROGRAM
CONTRACT NAS8-24017

SEPTEMBER 1970

Keith H. Hill
Robert O. Leighou
Duane L. Starner


Robert O. Leighou
Program Manager

Prepared For

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER ALABAMA 35812

Electronics Research Department
Martin Marietta Corporation
P. O. Box 179, Denver, Colorado 80201

FOREWORD

This supplement to the final report is presented in response to Paragraph III.2 of Exhibit A of Contract NAS8-24017.

CONTENTS

	<u>Page</u>
Foreword	ii
Contents	iii
Abstract	iv
I. Introduction	1
II. Basic Principles of Prime Implicant Generation	2
III. General Description of the Algorithm	7
IV. Data Preparation	9
V. Prime Implicant Generation	14
VI. Prime Implicant Selection	19
VII. Conclusions	21
<u>Figure</u>	
II-1 Flow Chart for Subcube Generation	5
II-2 Flow Chart for Subcube Vertex Generation	6
IV-1 Master Control Card	10
IV-2 MCARR Table	11
IV-3 DONT CARE Control Card	12
V-1 Prime Implicant Generation Flow Chart	15, 16, 17
VI-1 Prime Implicant Selection Flow Chart	20

ABSTRACT

The algorithm described is a fast algorithm for minimization of Boolean functions to a two level AND-OR form. The major advantages of this algorithm are that it is fast and that it does not require excessive storage capability. The speed and low storage requirement allows minimization of Boolean functions with a large number of variables such as 16. A 16 variable problem is too large to work manually and must be done a portion at a time using just a few of the variables. This is not a desirable method.

As an example of the performance of the algorithm, a 12 variable problem with approximately 50 percent of the possible combinations used, took 21.1 seconds of CPU time on a CDC 6500 computer including compilation, assembly and load time. Less than 65 K words of storage were required. From test problems with 8 through 12 variables, a projected CPU time for a 16 variable problem is 22 minutes and the storage remains constant because the algorithm was originally set up for the 16 variable problem.

This algorithm thus provides the capability for minimization of Boolean functions for large number of variables which were previously done poorly by manual methods and could not be done with a computer because of excessive time and storage requirements.

I. INTRODUCTION

The need for the algorithm that has been developed was generated as the result of development work on an addressable time division data system for NASA's Marshall Space Flight Center. This system is controlled by a sequence of addresses generated by a central unit. The method used to generate the addresses was to decode the state of word and frame counters contained in the central unit. The logic required for this decoding becomes quite extensive unless the address sequence is very simple. It should be noted that the address sequence used can also affect the logic required and also may require considerable effort to determine a useable sequence. In work just prior to this, a computer program was developed to determine the Boolean functions required to set an address shift register using the data system requirements. These requirements were: the words per frame; the frames per master frame, and the sampling rate per master frame for each data point. The Boolean functions generated by this program contained a large number of terms and needed extensive minimization in order to be practical.

The first step in development of the minimization algorithm was a literature search which revealed the early basic work of Quine ^{1,2} and McCluskey ³ and the later contribution of Carroll ⁴. The theorems developed by Carroll are presented and discussed in Section II since they were most useful in the development of this algorithm.

1. W. V. Quine, "The Problem of Simplifying Truth Functions", American Math Monthly, Vol. 59, pp. 521-533, October 1952.
2. W. V. Quine, "A Way to Simplify Truth Functions", American Math Monthly, Vol. 62, pp. 627-631, November 1955.
3. E. J. McCluskey, Jr, "Minimization of Boolean Functions", Bell System Technical Journal, Vol. 35, November 1956.
4. C. C. Carroll, "A Fast Algorithm for Boolean Function Minimization" AD 680305, Project Themis, Auburn University for Army Missile Command, Huntsville, Alabama, December 1968.

II. BASIC PRINCIPLES OF PRIME IMPLICANT GENERATION

This section describes the basic principles used in the algorithms as described in other sections. A thorough understanding of these principles is necessary for the comprehension of the algorithms.

The Boolean expression can be represented as a set of vertices of an n -cube for which the expression is true. n is the number of variables in the cube and 2^n is the total number of vertices in the cube. If the expression is represented by a sum of minterms, then each corresponds to a vertex of the n -cube. For ease of representation and computer manipulation each vertex or minterm can be specified by a binary number obtained by replacing complemented variables by 0's and uncomplemented variables by 1's (eg $\overline{A}BC = 101$). This binary number can then be converted to any other convenient base such as octal or decimal. The following discussions will use only binary to avoid confusion, but the concepts are valid independent of the number base used.

The principle of equation reduction is to combine the vertices of the function into groups called subcubes which can be represented by a single term containing fewer than n variables. Note that a subcube has 2^{n-x} vertices where x equals the number of variables in the subcube. The term for a single vertex has no variables, is a minterm and a zero-cube, i.e. 2^0 . A subcube of a function which is not contained in any other subcube of the function is called a prime implicant. The minimal solution for an expression in two level AND-OR form consists of a sum of prime implicants, however not all prime implicants need to be used. Since prime implicants are subcubes of the n -cube, it is necessary to understand some numerical properties of subcubes. The three theorems below were presented with proofs by C. C. Carroll¹. The proofs are not presented here, however discussions of each theorem is presented to aid in understanding the use of the theorems as presented in this report.

It is clear that for any subcube there is one vertex which has the largest binary value and one that has the smallest binary value. The operation " \wedge " between two vertices is defined as a bit by bit AND of the binary numbers (e.g. $1010 \wedge 0110 = 0010$). If two vertices v_1 and v_2 of an n -cube are such that $v_1 \wedge v_2 = v_1$, then this relationship is defined as $v_1 \leq v_2$ (e.g. $0101 \wedge 1101 = 0101$; therefore $0101 \leq 1101$). This can be thought of to mean v_1 is contained in v_2 .

1. C. C. Carroll, *ibid.*, p.1.

- Theorem 1: If $c \in C^n$, then $\min(c) \leq \max(c)$.

This theorem states that for any subcube, the minimum vertex ($\min(c)$) is contained in the maximum vertex ($\max(c)$).

- Theorem 2: $v \in C$ if $v \leq \max(c)$ and $\min(c) \leq v$

This theorem states that a vertex v of the n -cube C is an element of the subcube c if and only if v is contained in the maximum vertex $\max(c)$ and the minimum vertex $\min(c)$ is contained in v .

Theorem 2 proves that the minimum and maximum vertex of a subcube are sufficient to completely specify a subcube, and Theorem 1 provides a simple test to determine if two vertices determine a subcube. It is also apparent from theorem 1 that the maximum vertices for all subcubes with a common minimum vertex can be generated directly. This can be done by taking the 0's of $\min(c)$ and letting them take on all possible combinations of 1's and 0's, keeping the 1's of $\min(c)$ fixed. Similarly all vertices of a subcube can be generated by using theorem 2. Take all 0's of $\min(c)$ which correspond to 1's of $\max(c)$ and let them take on all combinations of 1's and 0's, keeping fixed the 1's and 0's of $\max(c)$ and $\min(c)$ which correspond.

An example of subcube generation with a common $\min(c)$:

Let $\min(c) = 01010$. the subcubes

are:	01010,	01010 (the vertex $\min(c)$)
	01010,	01011
	01010,	01110
	01010,	01111
	01010,	11010
	01010,	11011
	01010,	11110
	01010,	11111

An example of subcube vertex generation. Take the subcube 01010, 11011. The vertices of this subcube are

01010

01011

11010

11011

The computer implementation of the two generation processes are straightforward iterative procedures. For the subcube generator one starts with the first max (c), which is equal to 2^n-1 for the largest subcube. The remaining max (c)'s are obtained by subtracting binary numbers called RESULT, from 2^n-1 . RESULT takes on all binary values that have ZEROS in the positions corresponding to ONES of min (c). The RESULT values are generated in ascending order which generates subcubes in descending order. Figure II-1 is a Flow Chart of this process.

The generation of the vertices of the n-cube starts with min (c) as the first vertex. The complement of max (c) is bit by bit ORed with this vertex with a binary one being added to the result. Following the addition, a bit by bit OR with min (c) is performed followed by a bit by bit AND with max (c). This process continues until max (c) is reached. Figure II-2 is a flow chart of this process.

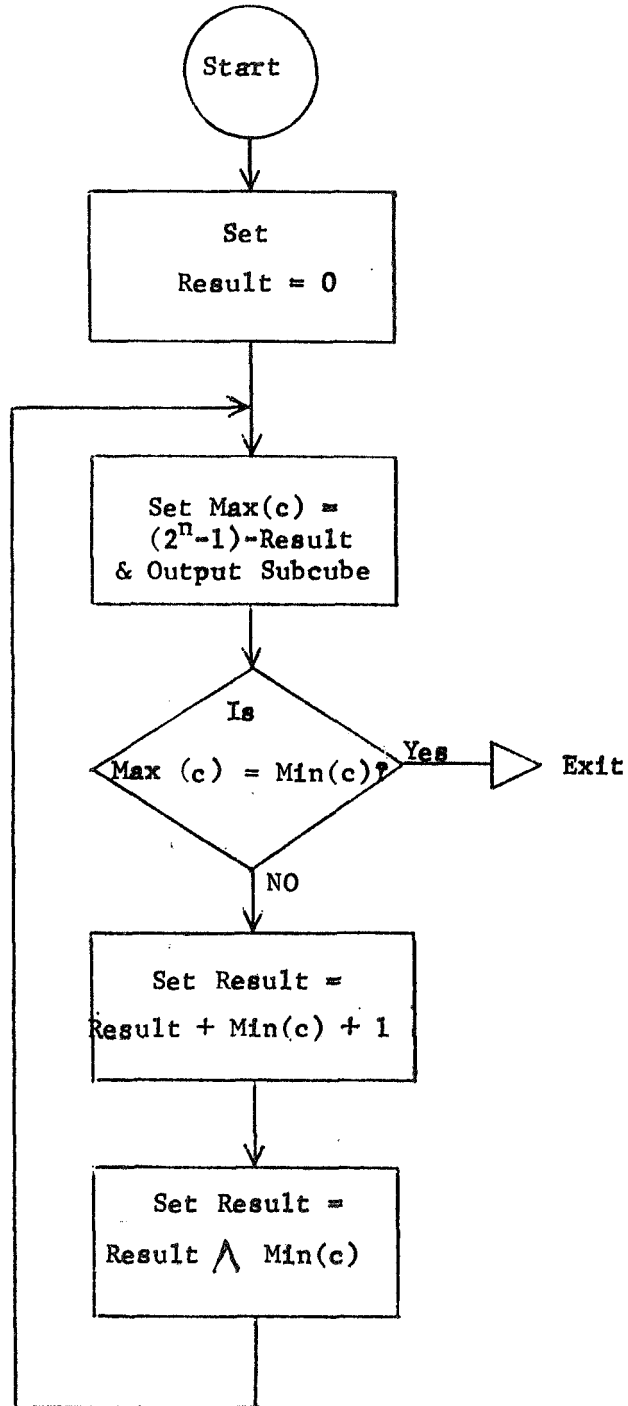


FIGURE II-1, FLOW CHART FOR SUBCUBE GENERATOR

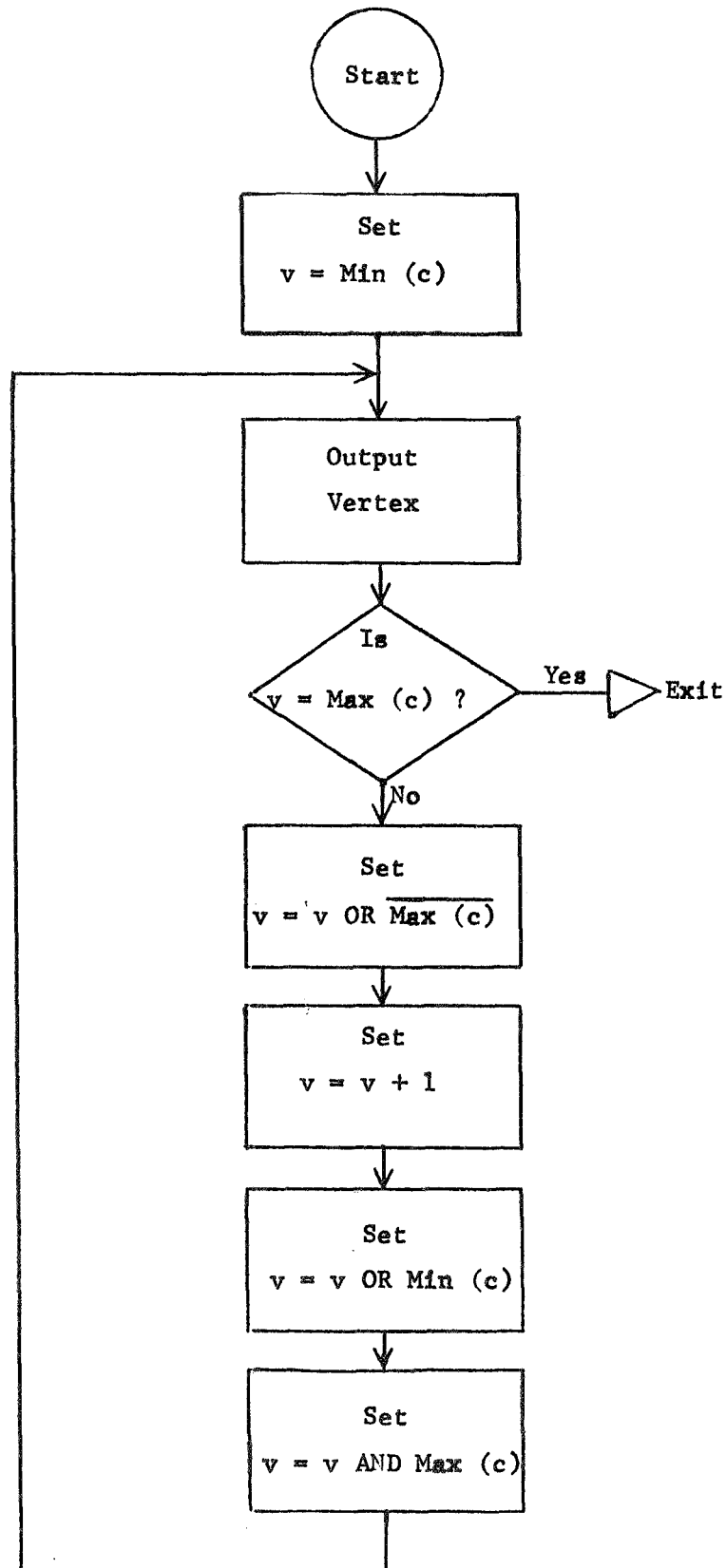


FIGURE II-2, FLOW CHART FOR SUBCUBE VERTEX GENERATION

III. GENERAL DESCRIPTION OF THE ALGORITHM

The minimization algorithm is based on the principles discussed in Section II. C. C. Carroll¹ also developed an algorithm of Boolean function minimization which was used as a basis for the development.

Briefly, the algorithm that has been developed has three parts. The first part is data preparation, the second part generates a number of prime implicants some of which may be redundant. The third part selects a non-redundant set of prime implicants that represent the original Boolean expression.

The data preparation part of the algorithm defines the size of the n-cube and lists the vertices that represent the Boolean expression and also those vertices which are DONT CARE. A DONT CARE vertex is the result of using feedback in a binary counter to cause it to recycle at non-binary rates. For instance, a four stage binary counter recycles every sixteen counts. However, with feedback, it can be made to recycle every 10 counts. Now if a Boolean function is to be used to express counts 8 and 9 (where 0 is the first count) the vertices 8 and 9 represent the Boolean expression and can be expressed by a one-cube (i.e. 2^1). However, counts 10 through 15 can never occur due to the feedback. The vertices corresponding to these counts are DONT CARE vertices. If these DONT CARE vertices are used, a three-cube (i.e. 2^3) containing 8 vertices can be used to represent the Boolean expression. The logic required to implement a three-cube in a 16 state map is one of the four variables. However, the logic to implement a one-cube is three of the four variables and is obviously more costly.

In the prime implicant generation, the algorithm searches the n-cube for the largest subcubes that will cover all the vertices that represent the Boolean function. This search begins by finding and retaining the largest subcube whose vertices are either contained in the function or are DONT CARE and whose lowest vertex is the lowest vertex of the function. This is a prime implicant. The algorithm then proceeds to find and retain all other subcubes that contain the vertices that represent the function (using DONT CAREs when it is advantageous) except that those subcubes whose vertices are contained in larger previously retained subcubes are not retained because they are not prime implicants. A number of things are done to reduce computer running time. Among the more important are: Subcubes and their vertices are calculated rather than performing an increment-by-one search and then testing for validity; and the search for subcubes is terminated when all vertices of the function are contained in the retained subcubes (prime implicants).

1. C. C. Carroll, *ibid.*, p 1.

The non-redundant prime implicant (PI) selection algorithm is fairly straightforward. First, all essential PIs are retained. This is easily accomplished since, during PI generation, the information indicating that a vertex is contained in only one PI and the identification of that PI was retained. The selection for a non-redundant set of PIs is continued by sorting the remaining non-essential PIs in the order of 1) vertices not covered by essential PIs, 2) the size of the PI subcube, 3) the order of selection. The rationale for this order is that 1) the PIs covering the most vertices contain the fewest redundant covered vertices, 2) the largest PI subcubes require the least logic for implementation, and 3) the PIs selected last may be bigger than those selected first and are at least the same size.

The algorithm outlined above has been tested on a number of problems and has always found the minimum normal form (MNF). This does not mean that it will always find the minimum normal form because we have no proof that the PI selection routine will always find the MNF. However, we have no proof it won't, either.

IV. DATA PREPARATION

The program prepares for the algorithm by first reading a control card. This card consists of 17 fields as shown in Figure IV-1. The size field controls the size of the Karnaugh Map (MCARR, Figure IV-2) used by the program. The remaining fields are used to control the interpretation of a term of an equation. The bit numbers which are active are inserted into the first fields with all other fields being zero. Thus for a four variable problem the first four fields are filled with 1, 2, 3, and 4.

The next card(s) are the cards containing the information about the DONT CARE (excluded) states. This card consists of six fields as shown in Figure IV-3. The first field is an end-of-data type indicator and is used only following all cards which contain data (of which there may be none). The second field contains the first DONT CARE state expressed in a decimal number. The third field contains the last DONT CARE state expressed in a decimal number. The fourth field contains the multiplier. The fifth field contains the first number to be multiplied. The sixth field contains the last number to be multiplied.

The data preparation phase of the program first initializes MCARR to zero using the size input to the program to determine where to stop. The program then uses the DONT CARE control cards to set the DONT CARE state in MCARR. The program uses the following calculation to determine the bits to set for each DONT CARE control card:

$$(\text{first state} + N) + (\text{multiplier}) (\text{Multiplier from} + M)$$

where $N = 0, 1, 2, 3, \dots$ and $M = 0, 1, 2, 3, \dots$ and when $(\text{first state} + N) = \text{last state}$, then m is incremented and $(\text{first state} + N)$ is set to $(\text{first state} + 0)$. After $(\text{multiplier from}) = (\text{multiplier to})$ the next card is processed.

At this point in the data preparation phase, MCARR contains no care states. The program then reads an equation term in the form:

$$S1 = Q1 Q2 Q3' Q4$$

$$Q1 Q2' Q3$$

The equation term may be placed in any card column but may not extend to the next card. There may not be more than one equation term per card.

Those bit numbers which are to be considered in the generation of subcubes by the equation interpreter.

SIZE OF MCARR TABLE

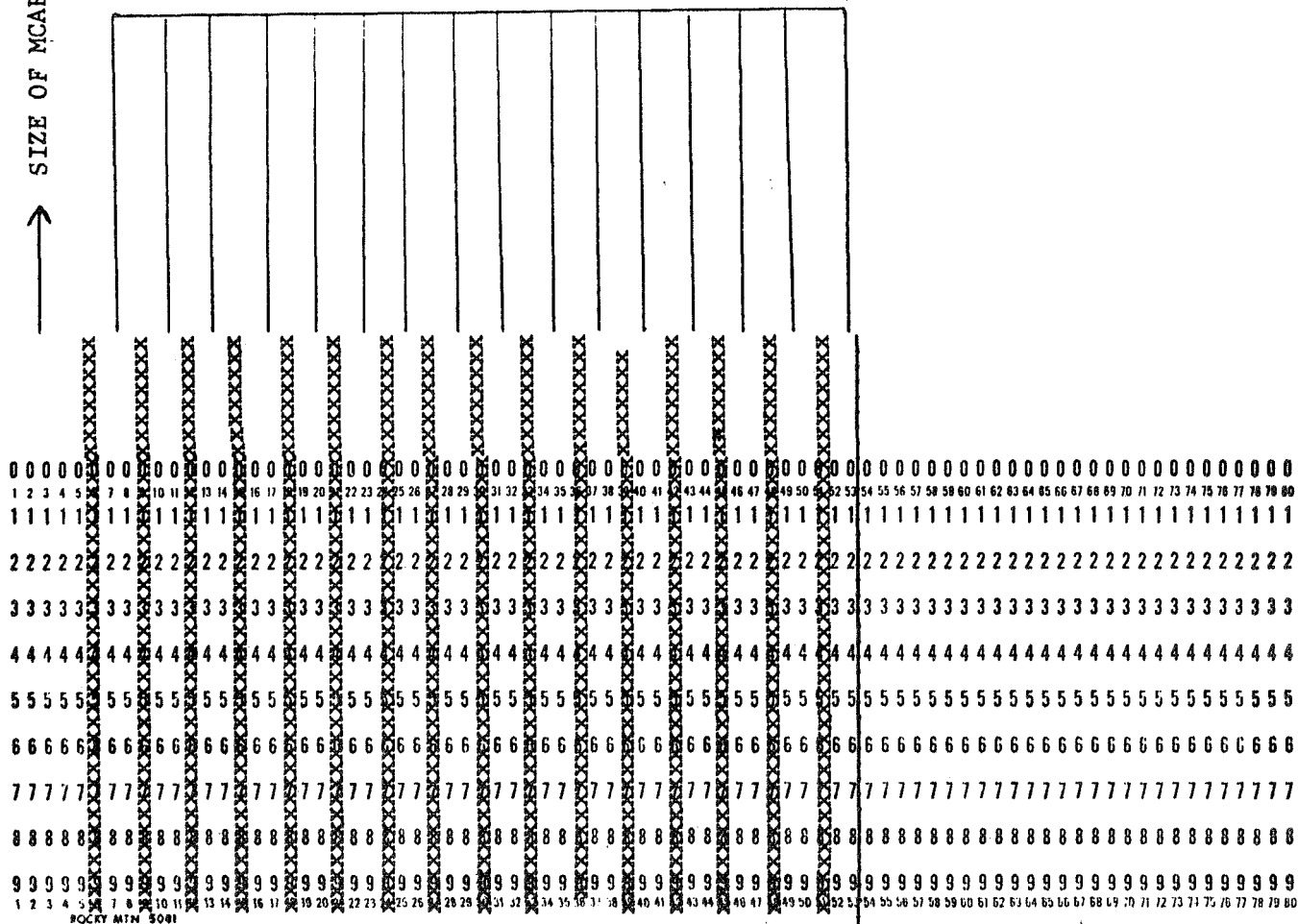


FIGURE IV-1, MASTER CONTROL CARD

MCARR TABLE

ENTRY 0	ENTRY 1	60 BIT WORDS
ENTRY 2	ENTRY 3	
ENTRY 4	ENTRY 5	
ENTRY 65532	ENTRY 65533	
ENTRY 65534	ENTRY 65535	

ENTRY DESCRIPTION

JF	CS	MPINUM	D	30 BIT ENTRY
----	----	--------	---	-----------------

JF - J FLAG - 1 octal digit - When set indicates that this is a good high vertex for a cube.

CS - CUBE SIZE - 2 octal digits - Set to the cube size which covers this vertex.

MPINUM - PRIME IMPLICANT NUMBER - 6 octal digits - This indicates the number of the cube which covers this vertex. If ZERO, the vertex has been covered by more than one cube. Used for essential prime implicant selection.

D - DESCRIPTOR - 1 octal digit - If set to ZERO indicates bit is ZERO.
 If set to ONE indicates bit is ONE.
 If set to TWO indicates bit is DONT CARE.
 If set to THREE indicates bit was ONE.
 and has now been covered; CS and MPINUM
 are used only in this state.

FIGURE IV-2, MCARR TABLE

The program reads the card and, using the bit numbers input in the master control card, interprets the term in the following manner. If all the bits called out in the master control card are contained in the equation term, then the bit pattern is used as a binary number pointing to that single care state. If all the bits called out in the master control card are not used in the term, then the unused bits are considered as X state bits and are taken through all possible states and all the resulting states are set into MCARR. The program then generates all the necessary prime implicants (as described in Section V) and selects the sufficient prime implicants (as described in Section VI). The program then determines by looking at the next card to be read if another equation is to be reduced.

The program determines the last care (ONE) bit set before it enters the prime implicant generation routine.

If another equation is to be reduced MCARR is initialized again and the same DONT CARE control cards are used to generate the DONT CARE states. If the card contains **** in the first four columns, the program terminates.

V. PRIME IMPLICANT GENERATION

The prime implicant generation routine, Figure V-1, determines the next non-ZERO low vertex (I) which is not greater than the last bit set to ONE. If the next I generated is greater than the last ONE bit, the program transfers to the prime implicant selection routine. This program calculates all upper vertices (J) which with I will form a prime implicant. All non-ZERO J's are flagged. The program uses the largest flagged J to form a cube (I,J). The size of the cube is calculated and saved.

For each vertex MCARR(K) of the cube, the following actions are taken:

- a. If MCARR(K) is a DONT CARE state (=2), the index K is saved in a list (E list) and the E bit counter is incremented.
- b. If MCARR(K) is ZERO, the J flag for this J is cleared and a new J is calculated to form a new (I,J) cube and all lists generated for the old cube are abandoned.
- c. If MCARR(K) is a CARE state (=1), the index K is saved in a list (L list) and the L count is incremented.
- d. If MCARR(K) is a COVERED CARE state (=3), the index K is saved in the E list and E list count is incremented and the size of the cube covering the vertex is examined. If the old cube size is greater than the size of the cube under examination, nothing further is done. If the cube under examination is larger than the old cube, the NONCNT counter is incremented.
- e. The index K is then changed to the value which points to the next vertex. If K is greater than J, the program begins to deal with the lists and counters generated in the MCARR(K) examinations, otherwise the examinations continue.

When a cube has passed all the MCARR(K) examinations, the cube is a prime implicant. For each element in the L list (i.e. CARE K's) the following operations are performed:

- a. The size of the current cube is placed in MCARR(L).
- b. The prime implicant number is placed in MCARR(L).
- c. The CARE state (=1) is changed to a COVERED CARE state (=3).
- d. The J flag is cleared for MCARR(L).

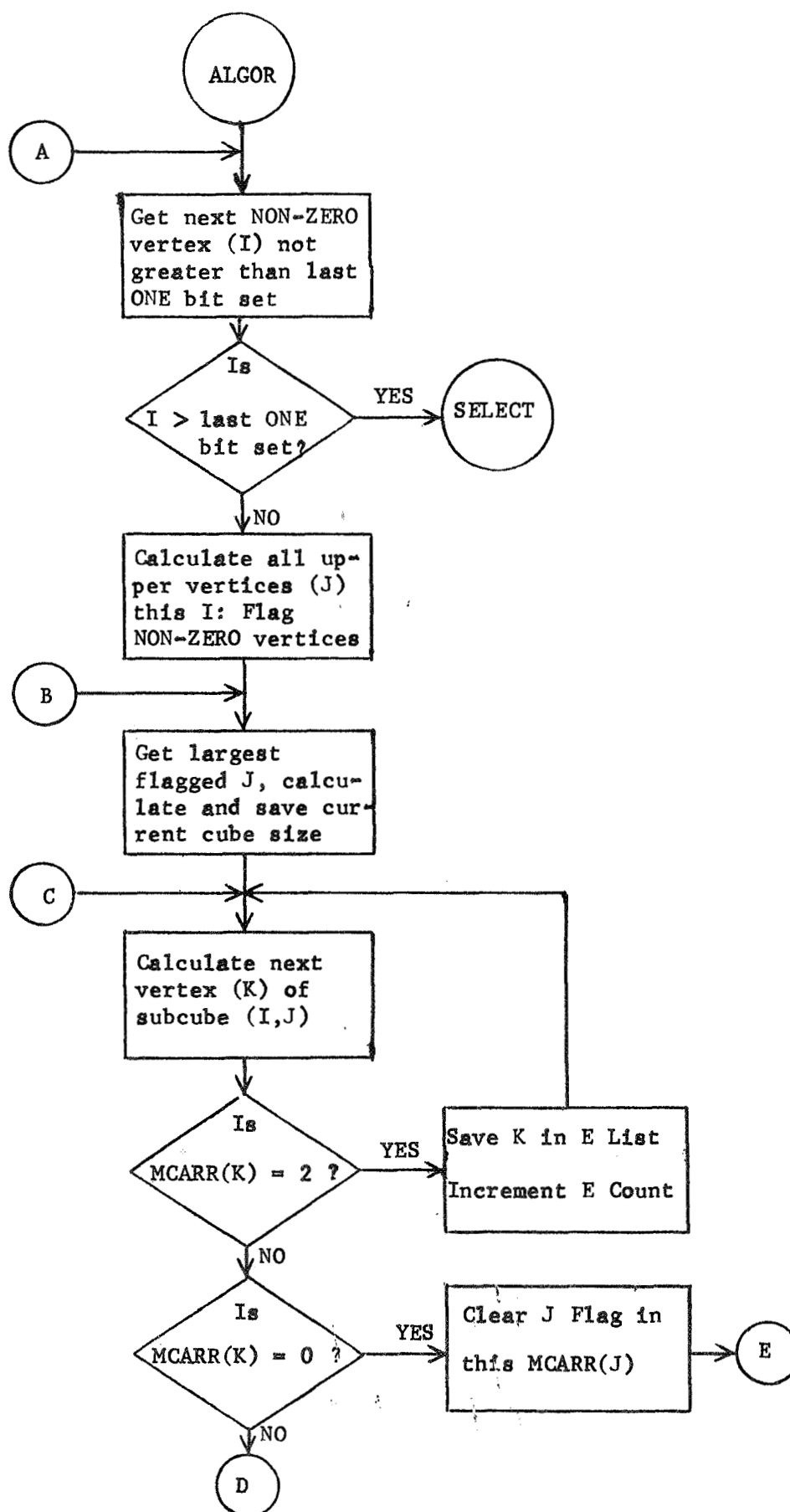


FIGURE V-1, PRIME IMPLICANT GENERATION FLOW CHART SHEET 1 of 3

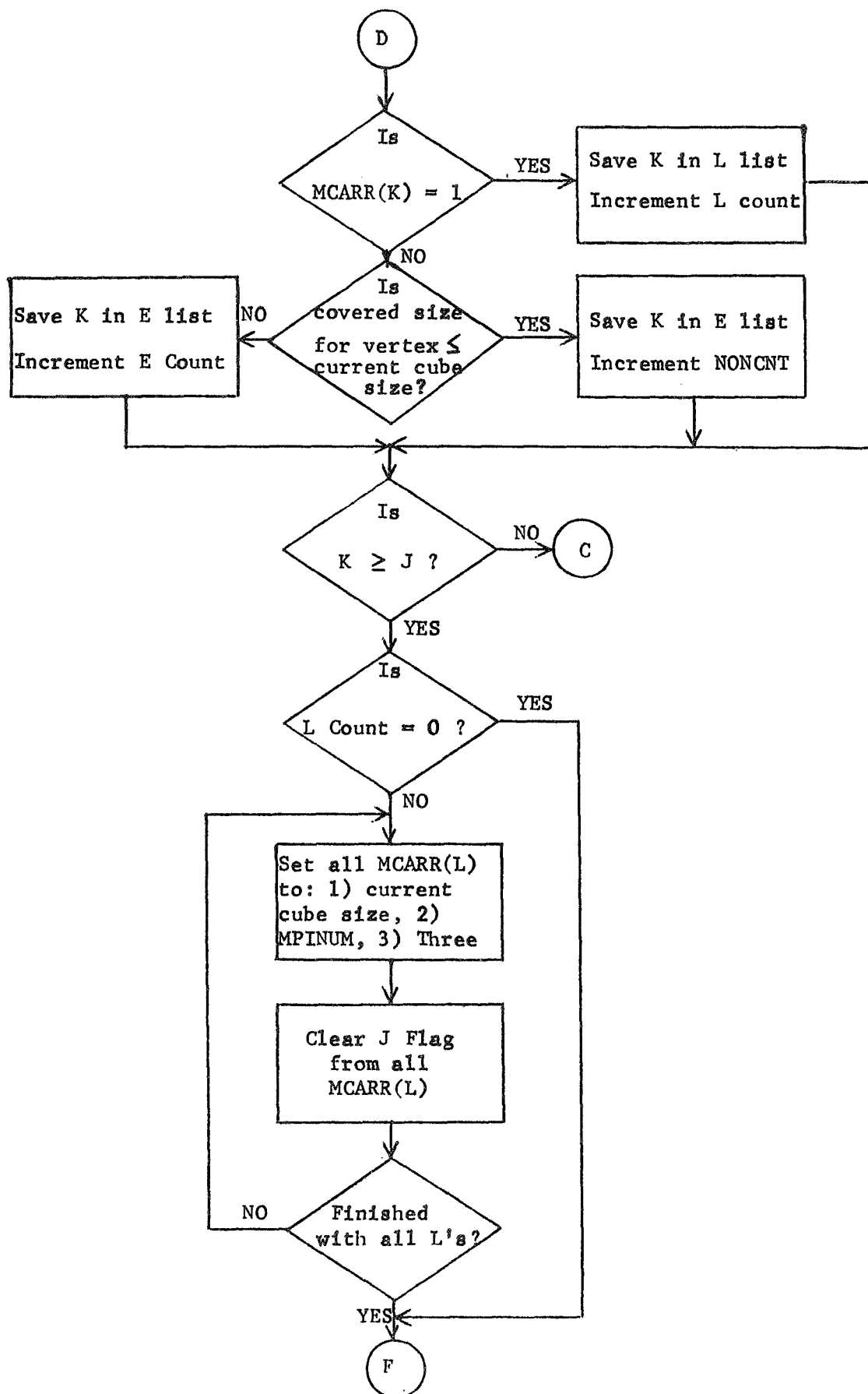


FIGURE V-1, PRIME IMPLICANT GENERATION FLOW CHART SHEET 2 of 3

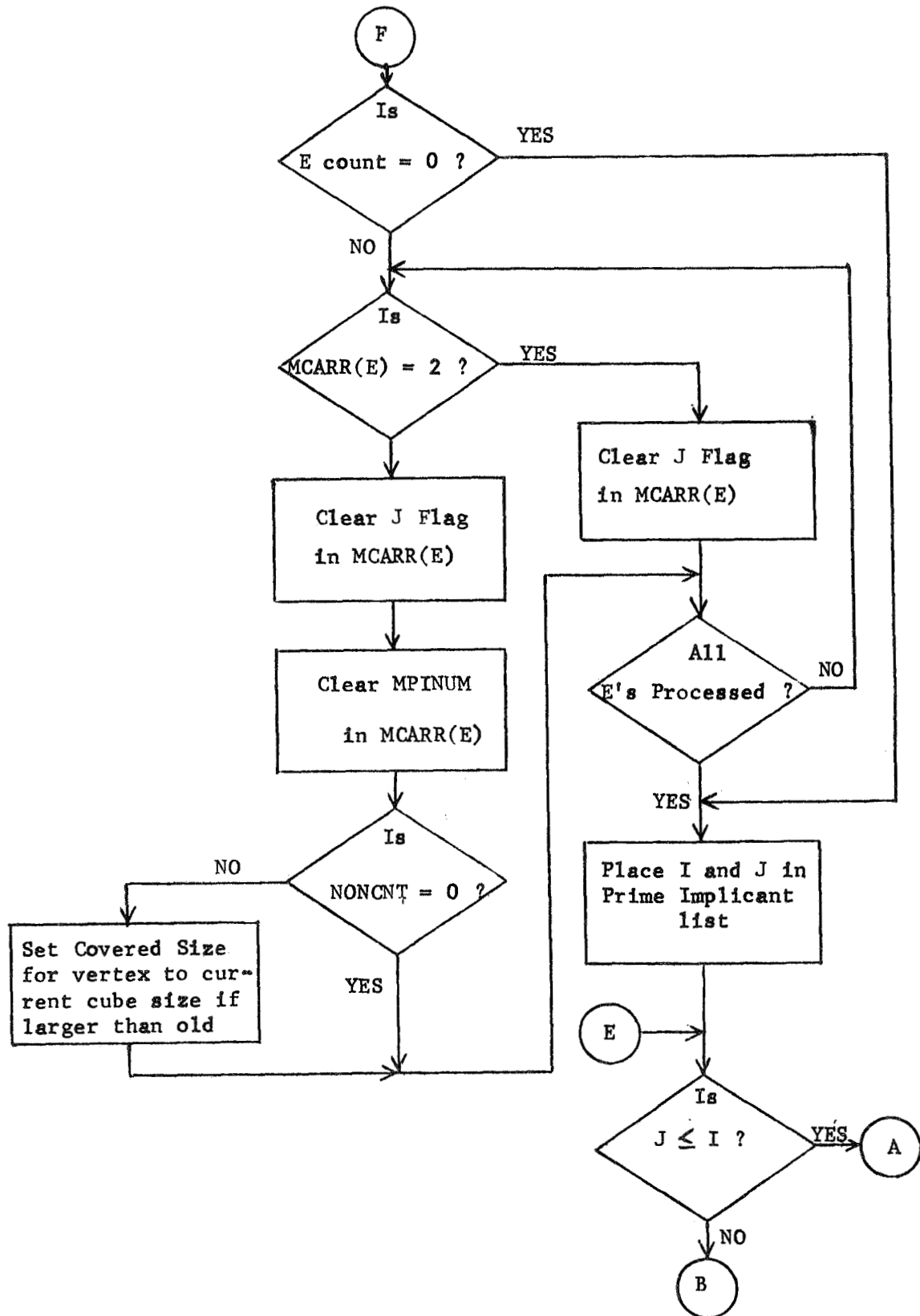


FIGURE V-1, PRIME IMPLICANT GENERATION FLOW CHART, SHEET 3 of 3

For each element in the E list (i.e. DONT CARE K's) the following operations are performed.

- a. The J flag in MCARR is cleared.
- b. If MCARR(E) is a DONT CARE state (=2), the next E is processed.
- c. The prime implicant number (I,J) in MCARR(E) is cleared.
- d. If NONCNT is not ZERO, each MCARR(E) is compared to determine if the old cube size is larger than the current cube size. If it is not, the old cube size is replaced with the current cube size, otherwise nothing happens.
- e. If NONCNT is ZERO, no action occurs.

After all the elements in the E list have been processed, the number (I,J) is placed in the prime implicant list. If J is less than or equal to I, the program returns to the beginning of the algorithm to find a new I, otherwise the programs determine a new J and repeats the above outline for the new (I,J) cube.

VI. PRIME IMPLICANT SELECTION

The prime implicant selection, Figure VI-1, first picks out the essential prime implicants. A prime implicant is considered essential when one or more CARE STATES are covered by only that prime implicant. This is done by examining each element of MCARR and if a prime implicant number is contained in the element, that prime implicant is essential. Each vertex of the essential prime implicant is set to ZERO.

For each non-essential prime implicant two comparison keys are generated, these are the size of the prime implicants and the order in which they are generated. The following actions are performed for each prime implicant:

- a. Build a major comparison key consisting of the number of CARE bits (=3) which are covered.
- b. Find the largest set of keys using them in order of 1) number of bits, 2) size of cube, and 3) order generated. If the major key of the selected prime implicant is ZERO, exit from program.
- c. Select this prime implicant.
- d. ZERO all vertices of the selected prime implicant.
- e. Return to step a.

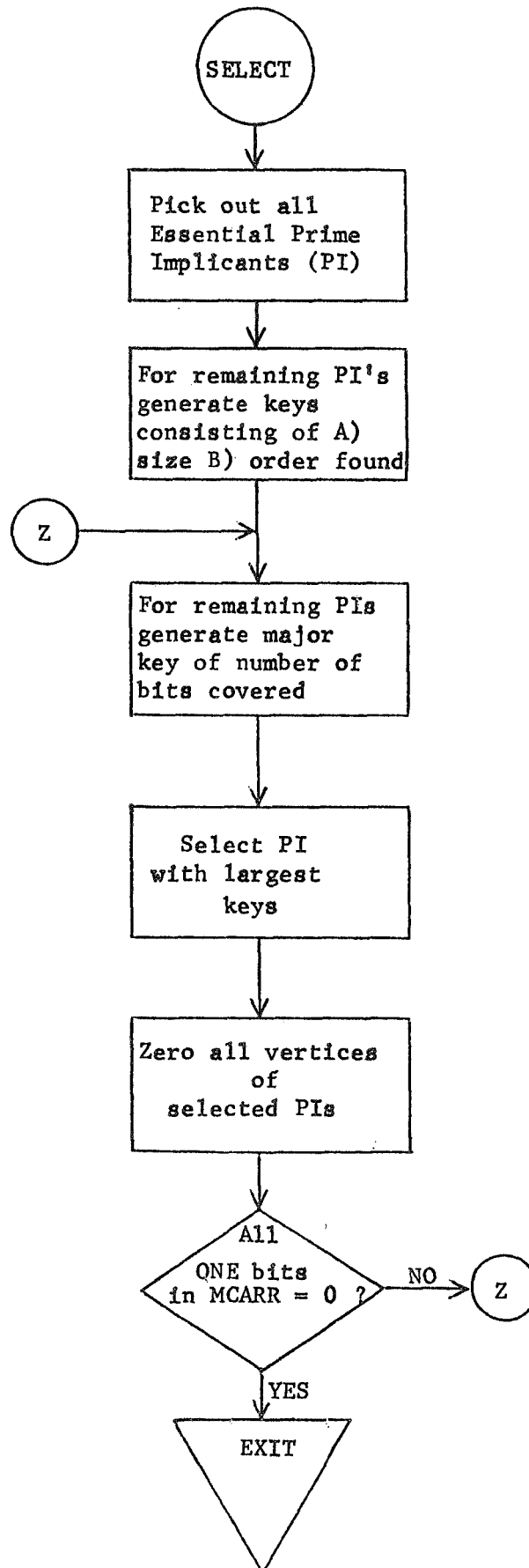


FIGURE VI-1, PRIME IMPLICANT SELECTION FLOW CHART

VII. CONCLUSIONS

The computer program that has been developed shows considerable promise in the minimization of Boolean functions. In particular, it provides the capability to minimize Boolean functions with up to 16 variables. In addition, it has the capability to make use of the forbidden states when the function is for non-binary systems. The minimization of Boolean functions during the checkout and test of the algorithm showed that the algorithm was indeed very fast, that it did not require excessive storage capability, and that it found the minimum two level AND-OR representation in all of the test problems. It may be that the algorithm will always find the minimum but the proof of this would require considerable effort. Since the algorithm will always provide a solution that is close to the minimum, this additional effort would not be warranted except for purely academic reasons.

The results of this program are so promising that the next step of minimization should be undertaken. In this step, the present algorithm should be modified so that multiple functions can be minimized. A very simple example of what could be done in this area is where three functions are implemented in a four variable (A,B,C,D) problem. If the functions are $\bar{A}D$, $\bar{A}\bar{B}D$ and $\bar{A}BD$, and a NAND-NOR logic family is used, the three functions can be implemented separately with a two-input NAND gate, two three-input NAND gates and three inverters. If the three functions are considered as a group, it can be done with two three-input NAND GATES, a two-input NOR gate and two inverters for a net savings of one inverter. For more complex problems, the savings can be much greater. This is particularly true for the set equations resulting from the format and address generation which tend to have similar terms in a number of the set equations.